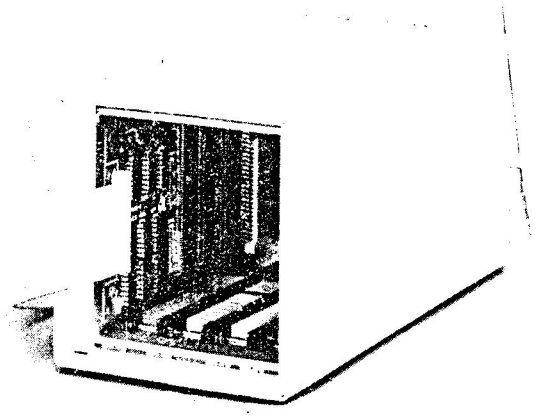
BEEBEX

General Purpose Eurocard Extension Unit
for the BBC Microcomputer 1 MHz Bus



Enclosed BEEBEX - power supply

- ★ **Attaches to the BBC Micro's 1 MHz bus**
- ★ **Allows CUBE Eurocards to be used as an extension to the BBC Micro**
- ★ **Allows access to an external additional 64kB memory map**

BEEBEX is probably the most versatile and comprehensive way of extending the BBC Microcomputer, because the CUBE range of Eurocards becomes available as hardware extensions using the 1 MHz bus interface.

The CUBE range includes:

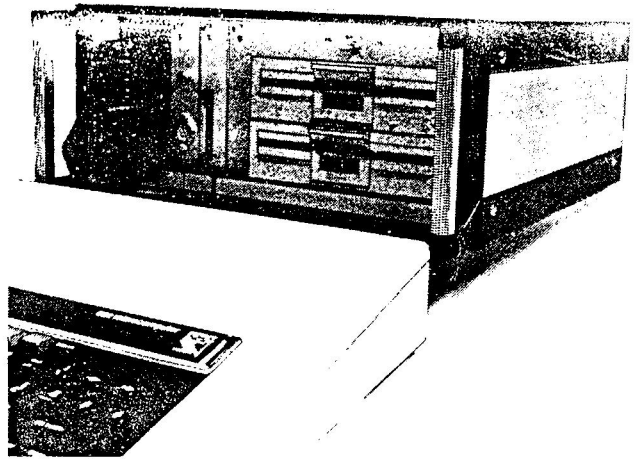
12 bit and 8 fast analog converters, DAC and ADC
Digital i/o
Serial i/o
Heavy-duty industrial opto-isolated i/o
Dynamic RAM memory and battery backed CMOS memory
VDU interface that provides full colour at high resolution of 512 x 256 pixels
In-circuit emulator
ROM emulator
Real-time clock
Liquid crystal display
Miniature printer

In addition, EuroBEEB is a single board computer which can support BBC BASIC, and can take over the function of the BBC after program development is complete.

Control Universal Ltd, 137 Ditton Walk, Cambridge CB5 8QF. Tel (0223) 244447

Control Universal Ltd
BEEBEX USER MANUAL

Hardware Expansion Unit for the BBC Microcomputer



★ HARDWARE EXTENSION TO THE BBC MICROCOMPUTER.
BEEBEX provides a convenient, robust and versatile means of adding further hardware facilities to the BBC micro.

★ ENTIRE CUBE RANGE CAN BE USED.
All CUBE Eurocard computer modules to be used to extend the hardware capabilities of the BBC Microcomputer.

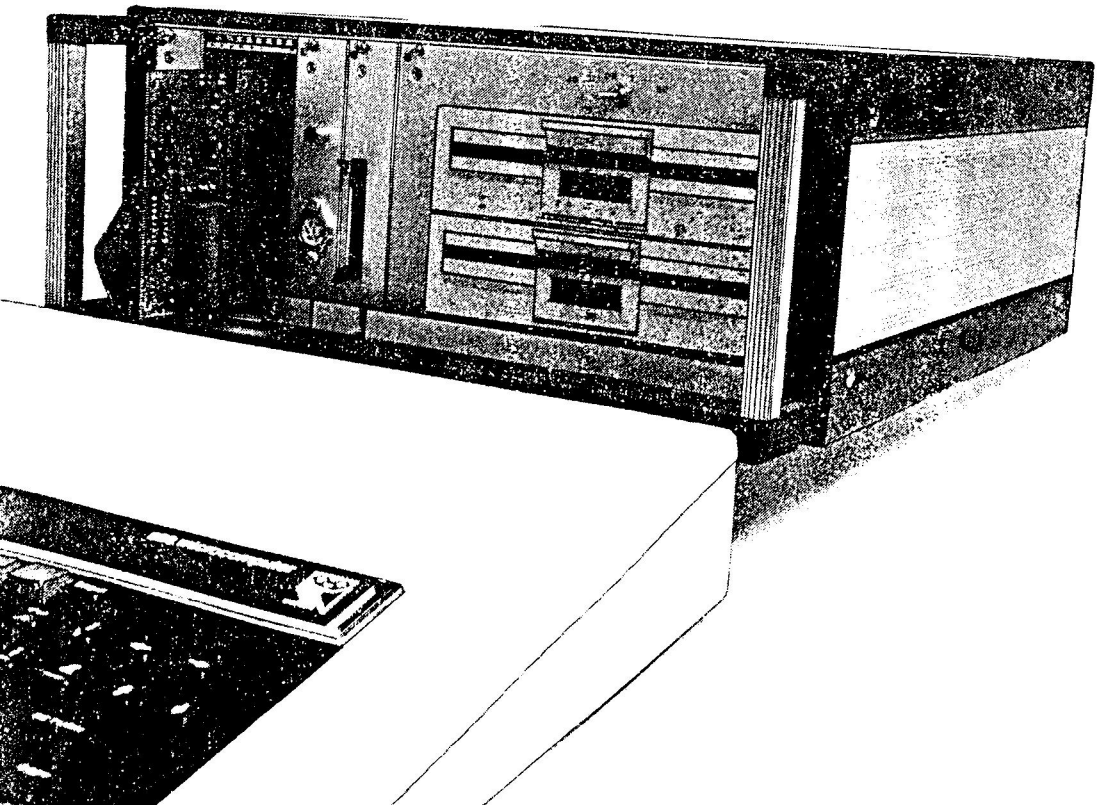
★ EXCEPTIONALLY EASY TO USE from the software point of view by use of a sideways ROM chip called *IO. See also separate data sheet on this product.

★ AUTOBOOTING. The BBC has an optional facility whereby a ROM (or EPROM) with address zero the Beebex memory map will be executed immediately upon switch-on. Such a ROM (or EPROM) can contain BASIC, machine code, or any other language for which there is an interpreter available in the machine.

★ EXAMPLES OF USEFUL EXTENSIONS include extra memory (including battery-backed CMOS), digital, analog and serial input/output channels and real-time calendar clock. CU-GRAPH, the true high resolution colour graphics display interface can be used, but does not come with software to be driven directly from the BBC operating system. This facility can be provided by the CUBE graphics terminal.

★ Industrial applications can use BEEBEX to develop control applications with the BBC machine, and then use EuroBEEB to replace the BBC in the target system.

BEEBEX USER MANUAL



SETTING UP THE BEEBEX

There are two versions of Beebex:-

- CUE2702 Beebex for rack mounting
- CUE2701 Beebex with 4 integral sockets for stand alone use

In addition there are a range of hardware and software options available

The two versions of Beebex are identical in use, the only difference being that the stand-alone version has four integral euro-sockets, while the rack version must plug into a Eurorack, which can have up to 16 sockets.

The 34 way cable supplied with Beebex is plugged into the 1MHz bus on the underside of the BBC micro.

As supplied, the address map on the Beebex is the standard 64KB of the typical 8 bit micro. To extend to 1MByte an extra latch chip must be fitted (see later). The whole of this 64KB is available to the user, so the CUBE modules to be used as BBC extensions can be set to any address demanded by the application. Note that a 64KB CU-DRAM memory card can be used in its entirity on its own, but if another device is to be used, the CU-DRAM must have one or more of its 4KB blocks deselected to make room for the extra device.

Future designs of CUBE modules will have 1 Megabyte addressing included as standard, which will allow 1 Megabyte of DRAM to be accessed simply by calling a megabyte address as shown later. Current designs are "paged". To call a particular CU-DRAM, a byte having a value in the range 0 to F (ie 0 to 16) is written to CU-DRAM hex address FFFF. The card having the code specified will be enabled and all other disabled. Note that any blocks disabled on the CU-DRAM map to accommodate other devices on the BEEBEX map must be disabled on all of the CU-DRAMs in use.

ACCESSING BEEBEX

Control Universal specify three methods of accessing Beebex.

*IO The easiest method is to use the sideways ROM called *IO. A separate publication describes this in detail, but for brief details see a section at the end of this manual.

Direct. The direct method of access is explained in the following pages, and has the advantage of being the fastest means of access.

OSBYTE. The BBC micro has, as one of its many strengths, a properly defined means of accessing expansion units. There are two gaps in the BBC memory map specifically for this purpose.

The gap reserved for controlling expansion units is named "Fred", and that for communicating with them, "Jim". (for no obvious reason).

These gaps, and "Sheila", for internal devices, are defined on page 436 of the BBC user manual, as follows:

Name	Memory address range	OSBYTE call Read	Write
FRED	&FC00-&FCFF	&92(146)	&93(147)
JIM	&FD00-&FD00	&94(148)	&95(149)
SHIELA	&FE00-&FEEF	&96(150)	&97(151)

Within the Fred i/o gap of 256 bytes, from hex FC00 to hex FCFF, allocations have already been made for specific purposes, such as the IEEE interface from hex FC20 to FC27. The gap reserved for Beebex is the "paging register", and is the last byte (hex FF) in Fred.

There are further bytes left free for the user to allocate, and one is employed by Beebex for the extended (1MB) addressing, and is the last but one byte, at hex FE.

To fully define a byte within a 1 Megabyte range, 20 bits are required, which can be broken down as follows:

1st 8 bits (lowbyte)	2nd 8 bits (midbyte)	3rd 4 bit (hinibble)
define byte within 256 bytes	define byte of 256 bytes in 64KB map	define map of 64KB within extended map of 1MB
address with JIM	Beebex low latch	Beebex high latch

Thus the Jim memory gap of 256 bytes exists in the Beebex memory map, and on the BBC micro map simultaneously. However, from the point of view of the BBC, these 256 bytes are at hex FD00 to hex FDFF, while on the Beebex they are at hex QXY00 to QXYFF. (note that a five digit hex number is required to define a number within a 1MB range).

The value Q is the hinibble number on the Beebex high latch, and the value XY is the midbyte number on the low latch. The low byte number on the Beebex is exactly the same as the low byte number on the BBC.

To set the midbyte and hinibble numbers the required value is written to the latches by the BBC at hex FCFF and FCFE respectively. However, to further ease this operation, the operating system provides standard functions, called OSBYTE calls, with rules for setting the parameters to be used.

WRITING TO BEEBEX - within 64KB map.

DIRECT METHOD, BASIC

write the value hex JK to address hex XYNM in the Beebex map

```
jim = &FD00          define communications gap in memory
lowlatch = &FCFF     define low latch, to which midbyte will be written
?lowlatch = &XY      specify midbyte
jim?&NM = &JK        write the value hex JK to lowbyte hex NM
```

Note that lowlatch need be defined only once for access to all the bytes in Jim, so a loop to transfer 256 bytes of data would set up lowlatch to start with, and then would loop round the instruction accessing Jim, thus:

```
jim = &FD00          define communications gap in memory
lowlatch = &FCFF     define low latch, to which midbyte will be written
?lowlatch = &XY      specify midbyte
mem = &GHIJ          specify start address of memory in BBC
FOR A = 0 to 255     set up 256 long loop
jim?&(A) = ? (mem+A) write the value read from memory
                     to lowbyte incremented from 0 to 256
NEXT
```

e.g. write the value decimal 127 (= hex 7F) to location hex 6C54 in the Beebex 64 KB map.

```
jim = &FD00
lowlatch = &FCFF
?lowlatch = &6C
jim?&54 = &7F (or = 127)
```

DIRECT METHOD, MACHINE CODE

```
LDA  #&XY          LDA  #&6C
STA  &FCFF         STA  &FCFF
LDA  #&JK          LDA  #&7F (or LDA #127)
STA  &FDNM         STA  &FD54
```

STANDARD OSBYTE (OPERATING SYSTEM) CALL METHOD

The direct method does not obey operating systems rules, but is only very marginally slower than a memory transfer within the BBC. The OSBYTE call method will be much slower, but is recommended when use of a second processor is envisaged.

OS call method in BASIC

OSBYTE is a standard Machine Operating System call but is not a reserved BASIC word, so must be defined before use in a BASIC program.

To write the value hex JK in the memory address hex XYNM in Beebex.

```
OSBYTE = &FFF4       OSBYTE exists at point in the OS ROM defined by &FFF4
A% = 147             defined required function, ie write to Fred,
                     to set low latch to desired value
X% = &FF             specify last byte in Fred, ie hex FF, which paging reg.
Y% = &XY             specify byte to be written as midbyte to Beebex low latch
CAL OSBYTE           execute OSBYTE function as defined
```



```
A% = 149             now write to Jim, to pass data byte from BBC to Beebex
X% = &NM             define lowbyte in Jim, to correspond to lowbyte in Beebex
Y% = &JK             specify value to be written
CALL OSBYTE          execute OSBYTE function as defined
```

example: write the value 127 (hex 7F) in the Beebex memory location hex 6C54.

```
OSBYTE = &FFF4
A% = 147
X% = &FF
Y% = &6C
CALL OSBYTE

A% = 149
X% = &54
Y% = 127
CALL OSBYTE
```

OS call method, machine code

```
method               example
LDA  #147            LDA  #147
LDX  #&FF            LDX  #&FF
LDY  #&XY            LDY  #&6C
JSR  &FFF4           JSR  &FFF4

LDA  #149            LDA  #149
LDX  #&NM            LDX  #&54
LDY  #&JK            LDY  #&7F (or LDA #127)
JSR  &FFF4           JSR  &FFF4
```

READING FROM BEEBEX

The choice of techniques is much the same as for writing to Beebex, with the same speed advantage of the "direct method", and with the same general principles for the OSBYTE call method.

Direct method, BASIC

To read the value in address hex XYNM in Beebex, and make it available as BASIC variable varl.

```
jim = &FD00          define communications gap in memory
lowlatch = &FCFF     define position of lowlatch in Fred
?lowlatch = &XY      define value of midbyte to be written in lowlatch
varl = jim?&NM       put in variable varl the value of the byte at &XYNM in Beebex
```

example: print the value in Beebex memory location hex 6C54.

```
jim = &FD00
lowlatch = &FCFF
?lowlatch = &6C
varl = jim?&54
PRINT varl
```

Read, direct method, machine code

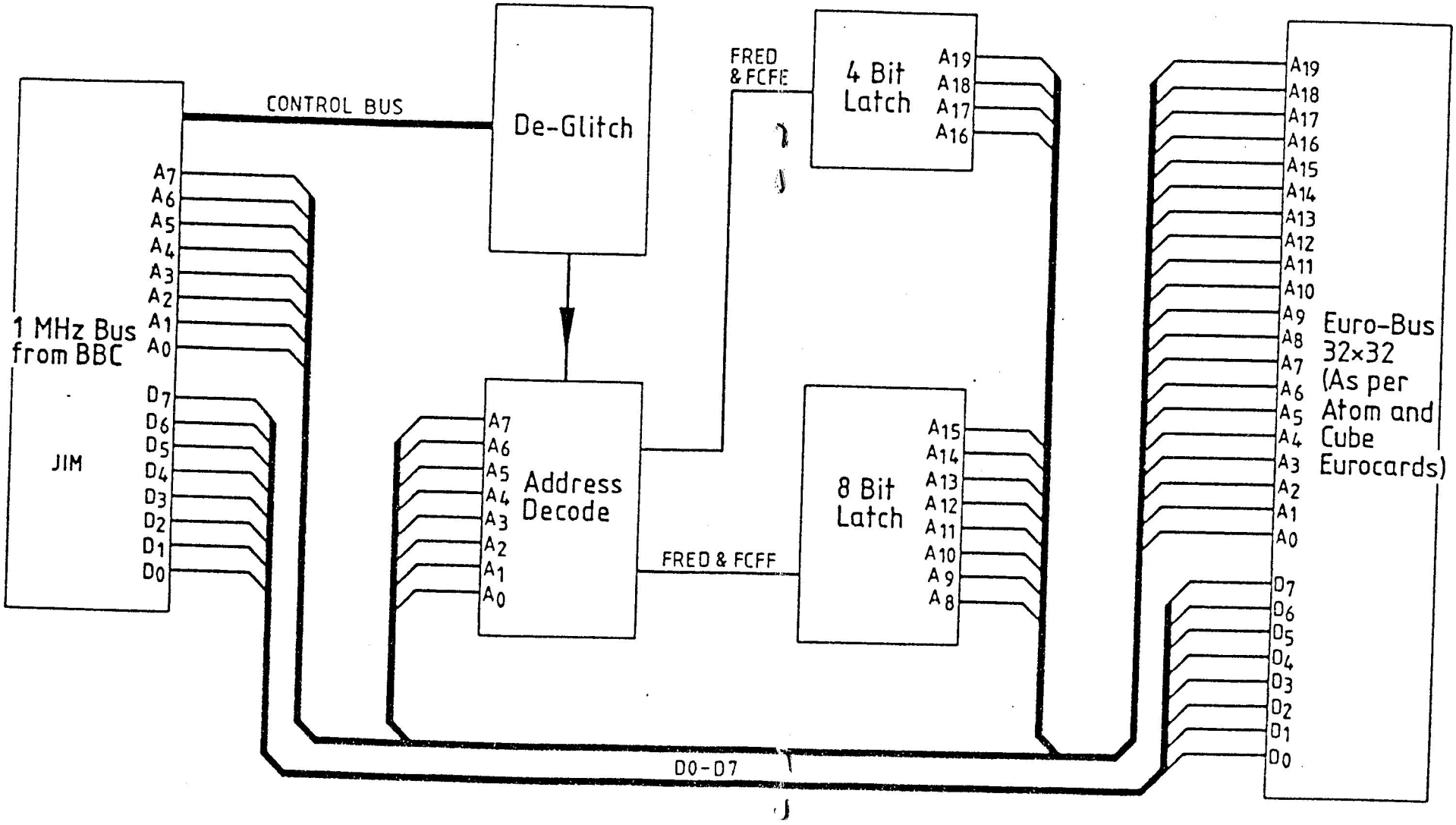
```
method               example
LDA  #&XY            LDA  #&6C      specify midbyte
STA  &FCFF           STA  &FCFF     write it to paging register in Fred
LDA  &FDNM           LDA  &FD54     read from specified byte in Jim
```

Value of specified byte is now in the accumulator of the cpu.

Read, OSBYTE method, BASIC

```
OSBYTE = &FFF4       define OSBYTE
A% = 147             define the required function, ie write to Fred,
                     to set lowlatch to desired value
X% = &FF             define last byte in Fred, ie paging register
Y% = &XY             set value of midbyte to put in Beebex lowlatch
CALL OSBYTE          execute, to set midbyte to hex XY

A% = 148             define OSBYTE function, ie read from Jim
X% = &NM             define lowbyte in Jim, corresponding to lowbyte in Beebex
varl = USR(OSBYTE)   execute read function
```


1 MHz Bus from BBC
JIM
CONTROL BUS
De-Glitch
A7
A6
A5
A4
A3
A2
A1
A0
D7
D6
D5
D4
D3
D2
D1
D0
Address Decode
FRED & FCFE
4 Bit Latch
A19
A18
A17
A16
FRED & FCFF
8 Bit Latch
A15
A14
A13
A12
A11
A10
A9
A8
D0-D7
Euro-Bus 32x32 (As per Atom and Cube Eurocards)

value of byte read will now be found in varI. eg. PRINT varI will display the value found. (alternatively, PRINT USER(OSBYTE) will print the value directly)

example: read into BASIC variable varI the value in Beebex hex 6C54.

```
OSBYTE = &FFF4
A% = 147
X% = &FF
Y% = &6C
CALL OSBYTE

A% = 148
X% = &54
varI = USR(OSBYTE)
PRINT varI
```

Read, OSBYTE method, machine code

Read in to accumulator the value in specified Beebex memory location.

Method	Example
Read from hex address XYNM	Read from hex address 6C54

```
LDA  #147          LDA  #147      define OSBYTE function as write to Fred
LDX  #&FF          LDX  #&FF      specify last byte in Fred
LDY  #&XY          LDY  #&6C      specify midbyte address in Beebex
JSR  &FFF4         JSR  &FFF4     execute Fred write function

LDA  #148          LDA  #148      define OSBYTE function as read from Jim
LDX  #&NM          LDX  #&54      specify lowbyte address in Jim
JSR  &FFF4         JSR  &FFF4     execute read from Jim
```

The data read from Beebex is now in the Y register of the CPU.

MEGABYTE ADDRESSING

Before the one megabyte address capability can be used the extra address latch chip must be fitted. This device is a 74LS173 and is fitted in the socket provided marked IC4. Take care to fit the right way round; the dot on the chip should be by the marked corner on the white ic marking on the pcb. The standard CUBE data bus (which is generally compatible with the Acorn Eurocard bus) how has added the extra address lines A16 to A19 for megabyte addressing. With the addition of the extra latch above, these appear on pins 15b to 12b respectively, on the CUBE 64 way DIN connector.

MEGABYTE WRITING

Direct method, megabyte write, BASIC.

To write the value hex JK to the address in Beebex hex QXYNM. (note five figure address for 1 MB map). The top four bits which define the block of 64 KB within the 1MB map is called here the "hinibble".

```
jim = &FD00          define communications gap in memory
lowlatch = &FCFF     define low latch to which midbyte will be written
?lowlatch = &XY      specify midbyte
hilatch = &FCFE      define high latch to which hinibble will be written.
?hillatch = &0Q      specify hinibble
jim?NM = &JK         write value hex JK to address hex QXYNM.
```

Direct method, megabyte write, machine code.

```
LDA  #&XY      load accumlator with midbyte
STA  &FCFF     store it in lowlatch (on last byte in Fred)
LDA  #&0Q      load accumulator with hinibble
STA  &FCFE     store it in hilatch (on last but one byte of Fred)
LDA  #JK       load accumulator with data value
STA  &FDNM     store it in the NM byte of Jim
```

Example: write the value decimal 127 (hex 7F) to address hex 76C54 in the Beebex 1MB map

BASIC	machine code

```
jim = &FD00          LDA  #&6C
lowlatch = &FCFF     STA  &FCFF
?lowlatch = &6C      LDA  #&07
hilatch = &FCFE      STA  &FCFE
?hilatch = &07       LAD  #&7F
jim?&54 = &7F (or = 127)  STA  &FD54
```

Megabyte write, OSBYTE method, BASIC

To write the value hex JK into the Beebex 1 MB map at address hex QXYNM.

```
OSBYTE = &FFF4     OSBYTE exists at the point in the OS ROM defined by &FFF4
A% = 147           define required function, ie. write to Fred
                   to set low latch to desired value of hex XY
X% = &FF           specify last byte of Fred, ie paging register
Y% = &XY           specify byte to be written as midbyte to Beebex low latch
CALL OSBYTE        execute OSBYTE function as defined

                   note that A% is still defined as write to Fred
X% = &FE           specify last but one byte of Fred, for hinibble latch
Y% = &0Q           specify which of 16 blocks of 64KB within 1MB map
CALL OSBYTE        execute OSBYTE function as defined

A% = 149           define OSBYTE function as write to Jim
X% = &NM           define byte address within Jim
Y% = &JK           specify data value to be written
CALL OSBYTE        execute function
```

Megabyte write, OS call method, machine code

```
LDA  #147          define function as write to Fred
LDX  #&FF          specify last byte of Fred
LDY  #&XY          specify midbyte to be written to lowlatch
JSR  #&FFF4        call OSBYTE

LDA  #147          define function as write to Fred
LDX  #&FE          specify last but one byte in Fred
LDY  #&0Q          specify hinibble to be written to high latch on Beebex
JSR  #&FFF4        call OSBYTE

LDA  #149          specify function as write to Jim
LDX  #&NM          specify byte within Jim to be written to
LDY  #&JK          load data to be written
JSR  #&FFF4        call OSBYTE and execute function
```

Example. Write the value decimal 127 (hex 7F) to location hex 76C54 in the Beebex 1MB memory map.

BASIC	machine code

```
OSBYTE = &FFF4     LDA  #147
A% = 147           LDX  #&FF
X% = &FF           LDY  #&6C
Y% = &6C           JSR  &FFF4
CALL OSBYTE

X% = &FE           LDA  #147
Y% = &07           LDA  #&FE
CALL OSBYTE        LDY  #&07
                   JSR  &FFF4

A% = 149           LDA  #149
X% = &54           LDX  #&54
Y% = 127           LDY  #&7F
CALL OSBYTE        JSR  &FFF4
```

MEGABYTE READING Direct method, megabyte read, BASIC

To read the value in megabyte address hex QXYNM in Beebex, and make it available as BASIC variable varl.

```
jim = &FD00          define communications gap in memory
lowlatch = &FCFF     define position of lowlatch in Fred
?lowlatch = &XY      define value of midbyte to be written in lowlatch
hilatch = &FCFE      define position of high latch in Fred
?hilatch = &0Q       define value of hinibble to be put in high latch
varl = jim?&NM       put in variable varl the value of the byte at &QXYNM in Beebex
```

example: print the value in Beebex megabyte memory location 76C54.

```
jim = &FD00
lowlatch = &FCFF
?lowlatch = &6C
hilatch = &FCFE
?hilatch = &0Q
varl = jim?&54
PRINT varl
```

Direct method, megabyte read, machine code

```
method          example
LDA  #&XY       LDA  #&6C     value of midbyte
STA  &FCFF      STA  &FCFF    write to paging register in Fred
LDA  #&0Q       LDA  #&07     value of hinibble
STA  &FCFE      STA  &FCFE    write to hinibble register in Fred
LDA  &FDNM      LDA  &FD54    read from lowbyte in Jim
```

Value of specified byte is now in the accumulator of the cpu.

Megabyte read, OSBYTE method, BASIC

```
OSBYTE = &FFF4       define OSBYTE
A% = 147             write to Fred, to set lowlatch to desired value
X% = &FF             define last byte in Fred, ie paging register
Y% = &XY             set value of midbyte to put in Beebex lowlatch
CALL OSBYTE          execute, to set midbyte to hex XY
                     A% still defined as write to Fred
X% = &FE             define last but one byte in Fred for hinibble latch
Y% = &0Q             define value of hinibble
CALL OSBYTE          execute function to set hinibble to hex 0Q
A% = 148             define OSBYTE function, ie read from Jim
X% = &NM             define lowbyte in Jim, corresponding to lowbyte in Beebex
varl = USR(OSBYTE)   execute read function
```

value of byte read will now be found in varl. eg. PRINT varl will display the value found. (alternatively, PRINT USR(OSBYTE) will print the value directly).

example: read to BASIC variable varl the value of Beebex 1MB address hex 76C54.

```
OSBYTE = &FFF4
A% = 147
X% = &FF
Y% = &6C
CALL OSBYTE
X% = &FE
Y% = &07
CALL OSBYTE
A% = 149
X% = &54
varl = USR(OSBYTE)
PRINT varl
```

Megabyte read, OSBYTE method, machine code

Read in to accumulator the value in specified Beebex memory location.

```
Method                          Example
Read from hex address XYNM      Read from hex address 6C54
LDA  #147                       LDA  #147    define OSBYTE function as write to Fred
LDX  #&FF                       LDX  #&FF    specify last byte in Fred
LDY  #&XY                       LDY  #&6C    specify midbyte address in Beebex
JSR  &FFF4                      JSR  &FFF4   execute Fred write function
LDA  #148                       LDA  #148    define OSBYTE function as read from Jim
LDX  #&NM                       LDX  #&54    specify lowbyte address in Jim
JSR  &FFF4                      JSR  &FFF4   execute read from Jim
```

The data read from Beebex is now in the Y register of the CPU.

ACCESSING BEEBEX WITH *10

*IO is a separate product with its own data sheet, and many more functions than just accessing Beebex. The following description covers only the Beebex-related topics of *IO.

*IO is a sideways ROM of the type now becoming increasingly popular for providing additional features on the BBC microcomputer. Other sideways ROMs include BASIC, Disk Filing System (DFS), Wordwise and Ultracalc.

*IO is a utility ROM, like the disk filing system ROM, and as such expects to be called from other sideways language ROMs. However, once called, vectors have been set up which direct all input and output calls to which ever i/o device has been specified to *IO.

The fundamental concept of *IO is that any area of memory outside the BBC computer is treated in the same way as a file on the file management system, ie like a disk file. Thus a named file can be opened, with a pointer within that file indicates where the current position is within that file, and with the ability to use the commands BPUT and BGET to write and read bytes from the Beebex memory map.

Example of the use of *IO.

In this example the BBC is connected via Beebex to the CUBAN-8 universal digital and analog interface. By writing an incrementing value to the 6522 (VIA) digital interface chip the on-board lead unit can be made to flash a binary representation of the numbers from 0 to 255. A time delay allows time to see the progressive changes.

```
 10 *IO                        call *IO
 20 pb=OPENUP"BUS &0E00"       BUS is a keyword recognised by *IO, and when used sets a
                               vector to the address next mentioned pb is the "handle" by
                               which port B, which exists at hex 0E00 on the CUBAN-8, can
                               be accessed.
 30 ddr=OPENUP"BUS &0E02"      the data direction register (ddr) exists at hex 0E02
 40 BPUT#ddr,&FF               write a byte to handle dddr, of value hex FF
 50 FOR A=0 TO 255             commence loop of 256 cycles
 60 BPUT#pb,A                  write a byte to handle pb, of value A
 70 PROCdelay                  call procedure named "delay"
 80 NEXT A                     end of loop
 90 GOTO 50                    run program again
100 DEF PROCdelay              define procedure named "delay"
110 FOR D=1 To 500:NEXT D      meaningless loop to cause delay
120 ENDPROC
```